Lecture 12, 21 September 2023

Linked lists

In [1]:
```
class Node:
    def __init__(self, v = None):
        self.value = v
        self.next = None
        return

    def isempty(self):
        if self.value == None:
            return(True)
        else:
            return(False)

    def append(self,v):   # append, recursive
        if self.isempty():
            self.value = v
        elif self.next == None:
            self.next = Node(v)
        else:
            self.next.append(v)
        return

    def appendi(self,v):   # append, iterative
        if self.isempty():
            self.value = v
            return

        temp = self
        while temp.next != None:
            temp = temp.next

        temp.next = Node(v)
        return

    def insert(self,v):
        if self.isempty():
            self.value = v
            return

        newnode = Node(v)

        # Exchange values in self and newnode
        (self.value, newnode.value) = (newnode.value, self.value)

        # Switch links
        (self.next, newnode.next) = (newnode, self.next)

        return

    def delete(self,v):   # delete, recursive
        if self.isempty():
            return

        if self.value == v:
            self.value = None
            if self.next != None:
                self.value = self.next.value
                self.next = self.next.next
            return
        else:
            if self.next != None:
                self.next.delete(v)
                if self.next.value == None:
                    self.next = None
        return

    def __str__(self):
        # Iteratively create a Python list from linked list
        # and convert that to a string
        selflist = []
        if self.isempty():
            return(str(selflist))

        temp = self
        selflist.append(temp.value)

        while temp.next != None:
          temp = temp.next
          selflist.append(temp.value)

        return(str(selflist))
```

In [2]:
```
l = Node()
l.append(5)
print(l)
```

```
[5]
```

In [3]:
```
l.appendi(7)
print(l)
```

```
[5, 7]
```

In [4]:
```
l.append(9)
print(l)
```

```
[5, 7, 9]
```

Rewrite `__str__` using `_tolist` that recursively constructs a Python list from the linked list

- Note, convention to prefix "hidden" function names with `_`

In [5]:
```
class Node:
    def __init__(self, v = None):
        self.value = v
        self.next = None
        return

    def isempty(self):
        if self.value == None:
            return(True)
        else:
            return(False)

    def append(self,v):   # append, recursive
        if self.isempty():
            self.value = v
        elif self.next == None:
            self.next = Node(v)
        else:
            self.next.append(v)
        return

    def appendi(self,v):   # append, iterative
        if self.isempty():
            self.value = v
            return

        temp = self
        while temp.next != None:
            temp = temp.next

        temp.next = Node(v)
        return

    def insert(self,v):
        if self.isempty():
            self.value = v
            return

        newnode = Node(v)

        # Exchange values in self and newnode
        (self.value, newnode.value) = (newnode.value, self.value)

        # Switch links
        (self.next, newnode.next) = (newnode, self.next)

        return

    def delete(self,v):   # delete, recursive
        if self.isempty():
            return

        if self.value == v:
            self.value = None
            if self.next != None:
                self.value = self.next.value
                self.next = self.next.next
            return
        else:
            if self.next != None:
                self.next.delete(v)
                if self.next.value == None:
                    self.next = None
        return

    def _tolist(self):
        # Recursively construct a Python list from linked list
        if self.isempty():
            return([])
        elif self.next == None:
            return([self.value])
        else:
            return([self.value] + self.next._tolist())

    def __str__(self):
        # Convert Python list representation of linked list to string
        return(str(self._tolist()))
```

In [6]:
```
l = Node()
l.append(5)
print(l)
```

```
[5]
```

In [7]:
```
l.appendi(7)
print(l)
```

```
[5, 7]
```

In [8]:
```
l.delete(7)
```

In [9]:
```
print(l)
```

```
[5]
```

Some performance measurement

In [10]:
```
import time
```

Insert items at the start of a linked list, multiples of 10^5, linear blowup

In [11]:
```
for i in range(1,5):
    l1 = Node()
    start = time.perf_counter()
    for j in range(i*100000):
        l1.insert(j)
    elapsed = time.perf_counter() - start
    print(i*100000,elapsed)
```

```
100000 0.4618713259987999
200000 1.163837590996991
300000 1.583010435999313
400000 2.245697131998895
```

Insert items at the start of a Python list, multiples of 5×10^4, quadratic blowup

In [12]:
```
for i in range(1,5):
    l2 = []
    start = time.perf_counter()
    for j in range(i*50000):
        l2.insert(0,j)
    elapsed = time.perf_counter() - start
    print(i*50000,elapsed)
```

```
50000 1.6438657019971288
100000 6.038590096999542
150000 11.514449295005761
200000 18.64402558000438
```